前后端联调方案探索与应用

吕伟@美团点评技术工程部高级工程师

https://github.com/ysmood

Vane 舵

http://vane.sankuai.com (仅内内网)

vane |veɪn|

noun

① (blade of machine, windmill) 叶片 yèpiàn

② = weathervane

③ (flat part of feather) 翎 líng

④ (fin or plate) (of torpedo) 舵 duò (of arrow) 箭翎 jiànlíng

你是否碰到过下面这些情况

开发个 API 怎么这么慢?

写单测好麻烦!

那前端一天要来问十几次 API 用法!

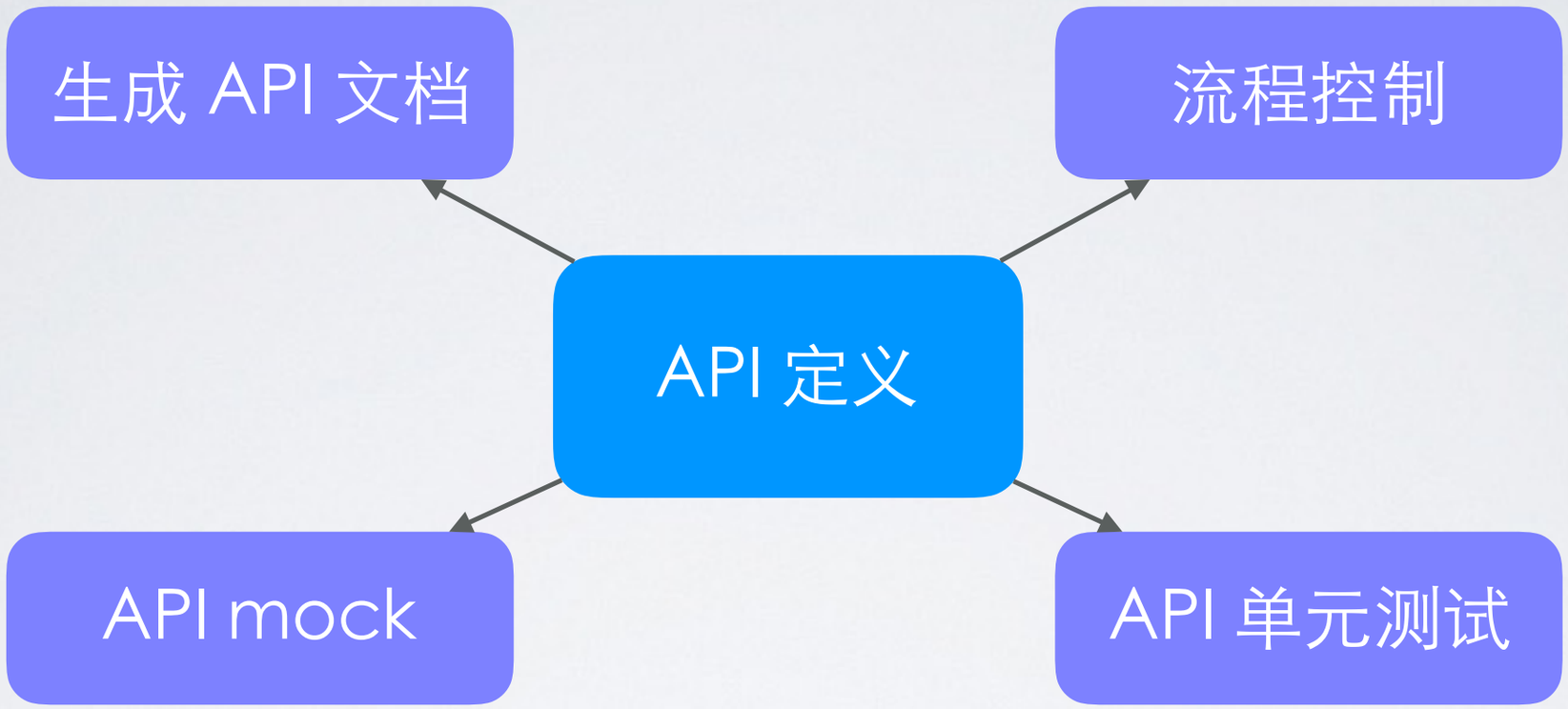
API 怎么老是变啊!

wiki 写起来好费时!

从解耦说起

- 不论前端还是后端都会有依赖
- 依赖的开发速度难以协调
- 让开发环境接近完美沙盒

Vane 想做的事



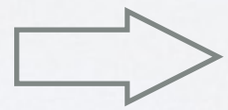
只需要定义一次 API 即可自动生成 API 文档， mock 假数据，单元测试 API，以及流程控制

API 定义

adoc (api documentation)

```
1
2  ## @url /items/{id}{?limit}
3
4  我们可以写任意的 markdown 来说明这个 API
5
6  ## @case
7
8  ### @response
9
10 - @body Hello World!
```

markdown 超集



示例

路径 /items

我们可以写任意的 markdown 来说明这个 API

用例

返回

- Hello World!

渲染成文档

Type 和 模糊匹配

````
- @body

  返回的数据类型符合下面的定义：

  ```type
  {
    id: int(0, 100)

    name: string()

    info: {
        colors: [
            cases("red", "green")
        ]
    }
  }
  ```
````

简化的 json schema 语法糖

支持 js 生成数据

```
- @body

  返回的数据类型符合下面的定义：

  ```js
  function main () {
      return [1, 2, 3].map(i => {
          id: i,
          value: i * i
      });
  }
  ```
```

安全控制

browser 端通过 web worker

node 通过 subprocess 和 vm

更多

```
- @request

  请求符合下面的定义：

  ```thrift
    service Calculator extends shared.SharedService {

      void ping(),

      i32 add(1:i32 num1, 2:i32 num2),

      oneway void zip()

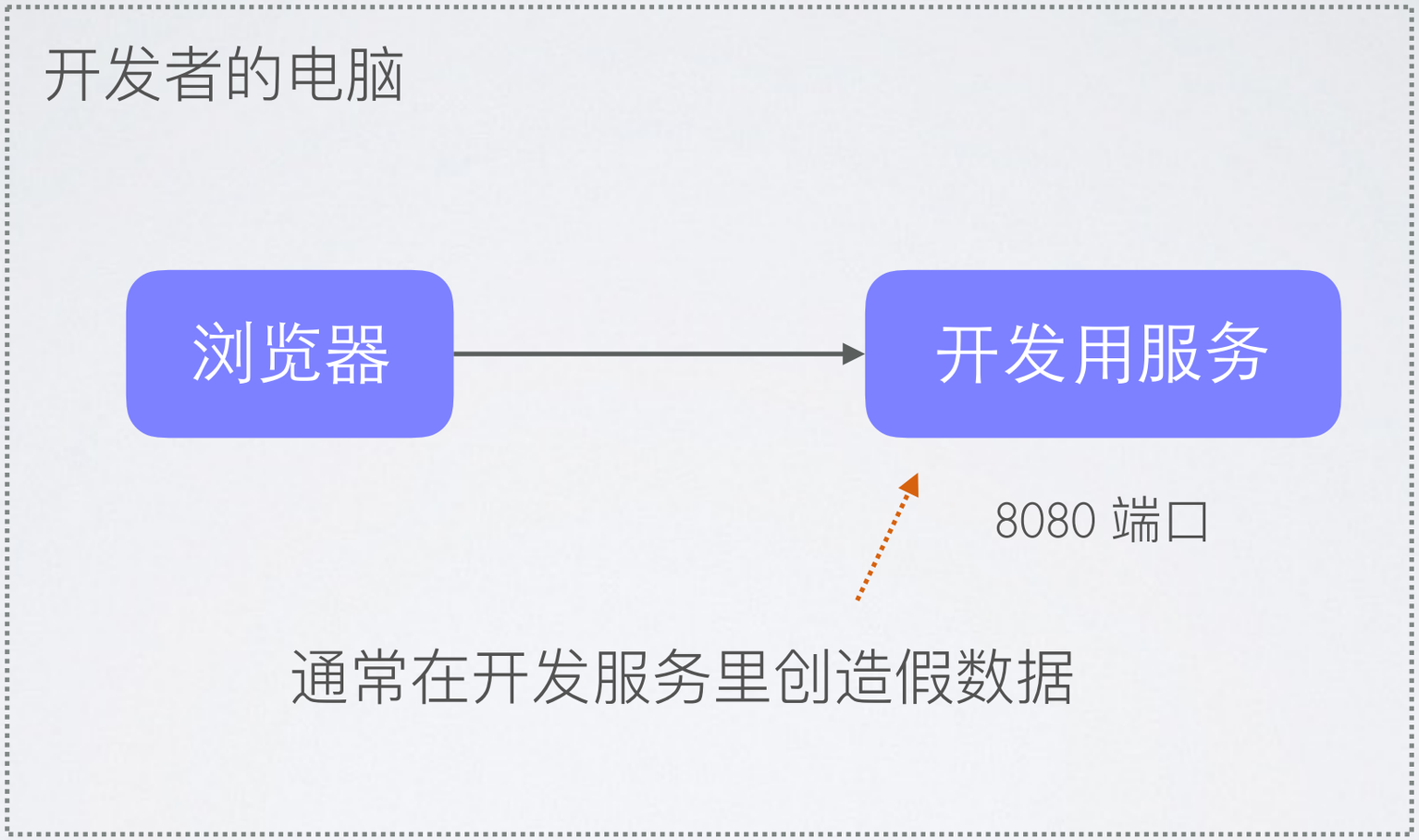
    }
  ```
```

几乎可以根据实际情况随意扩展

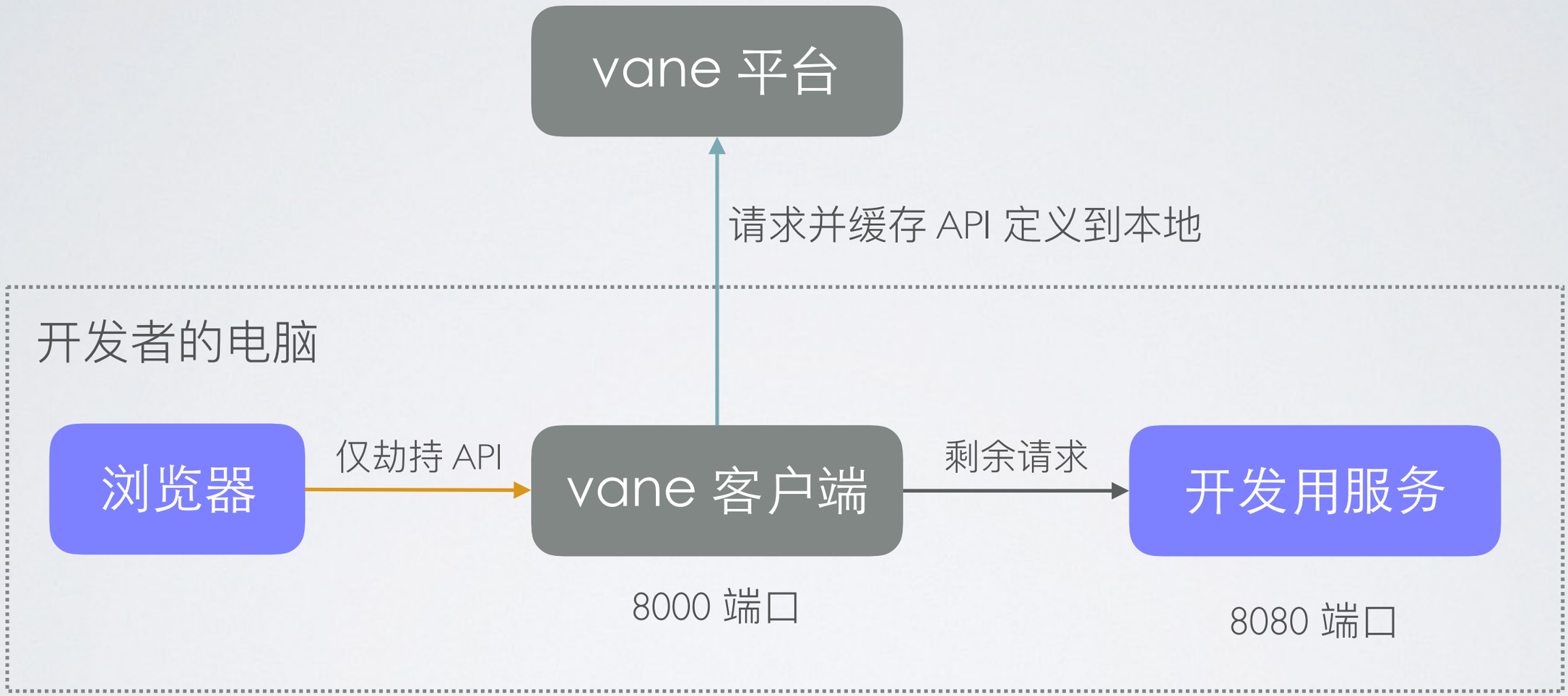
沙盒 => 客户端

- 类似于 github 和 git，vane 是 CS 架构的
- 即使离线状态 vane 也能正常使用
- 将运算压力转移到了各个用户端

一般的 mock 开发模式

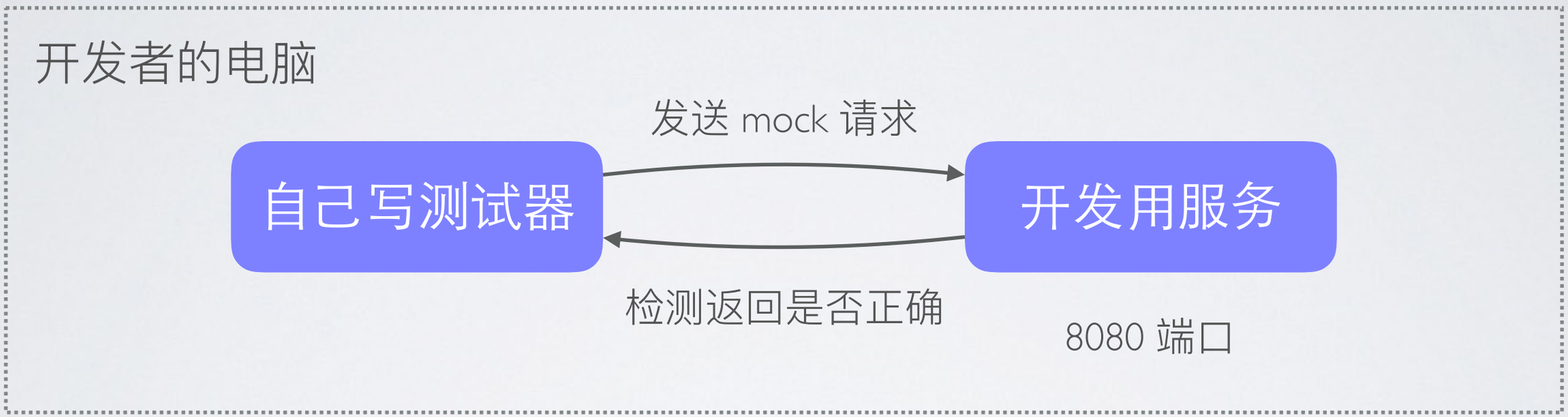


无痛 mock 接入



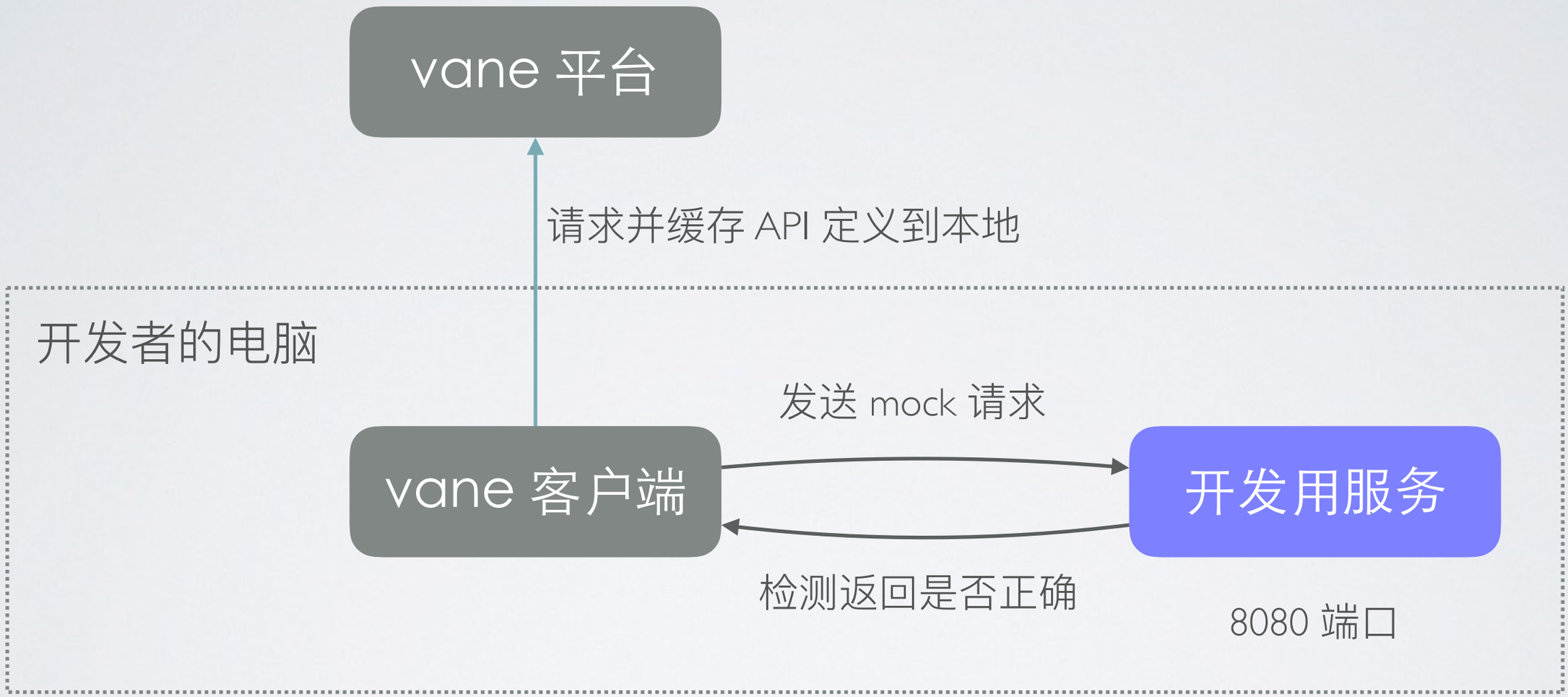
类似中间人攻击，vane 客户端的接入完全透明无痛

一般的开发服务 API 测试



需要费时间自己写测试，甚至费时去调试测试代码本身

API 测试接入


vane 平台
请求并缓存 API 定义到本地
开发者的电脑
发送 mock 请求
vane 客户端
开发用服务
检测返回是否正确
8080 端口

网页端已实现的功能

- 项目管理
- API 编辑
- 项目 Token
- API 变更历史
- API 搜索
- 成员管理
- API 变更通知

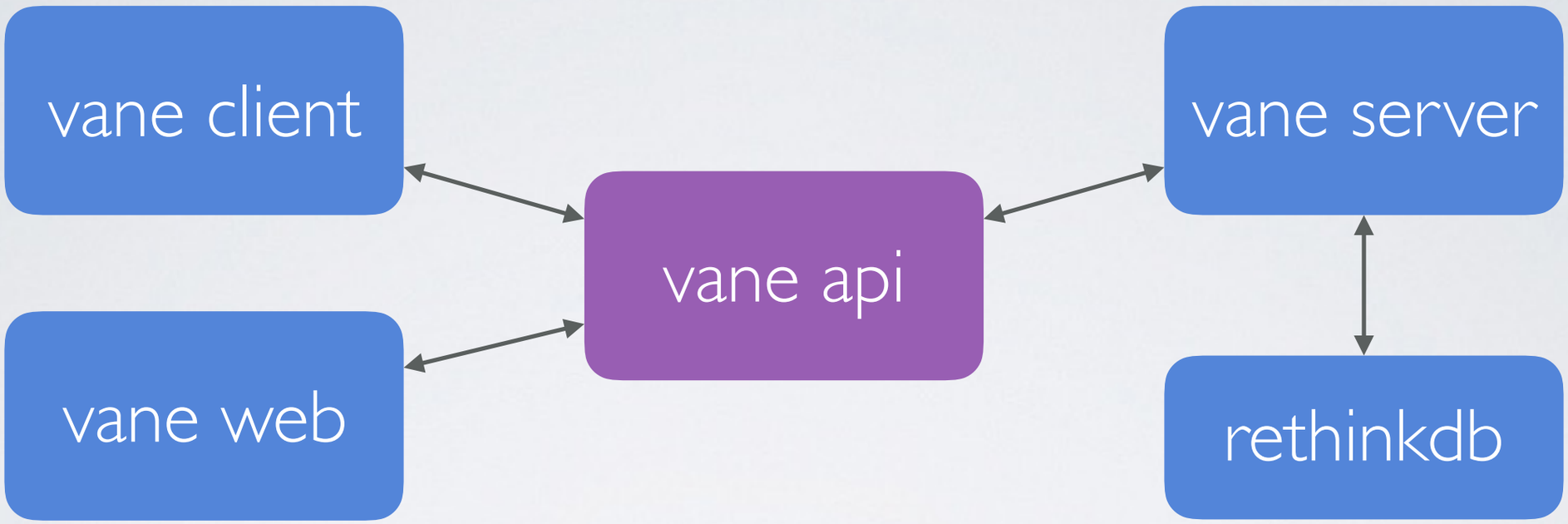
客户已实现的功能

- 实时获取 API 定义的变更
- 离线使用 API 定义
- 假数据代理
- 单元测试后端服务
- 作为库来使用

技术栈

- 纯 JS 技术栈，从自动化工具，到数据库操作，到后端逻辑，再到客户端实现
- 前端 React + Babel + Webpack
- 后端 Nodes + Rethinkdb

模块基本构成



adoc　nisp　noflow　junit　nokit

nofs　yaku　mx-fe-bone　……

nisp

- 一个非图灵完备的 lisp-0 语言
- 较 RPC 更为抽象灵活的一种思路
- 基于权限的语言，初始不含任何功能
- 项目地址：github.com/ysmood/noflow

```
var nisp = require("nisp");

var env = {
    do: require("nisp/lib/do"),
    set: require("nisp/lib/set"),
    get: require("nisp/lib/get"),
    if: require("nisp/lib/if")
};

var expresses = ["do",
    ["set", "a", ["if", false, 10, 20]],
    ["get", "a"]
];

nisp(expresses, env); // => 20
```

```
var nisp = require("nisp");
var plainFn = require("nisp/lib/plainFn");

var env = {
    session: { isAdmin: false },
    "+": plainFn(function (a, b) {
        if (!env.session.isAdmin) throw Error("permission not allowed");
        return a + b;
    })
};

var expresses = ["+", 1, 2];

nisp(expresses, env); // => Error
```

yaku

异步调试，稳定性，扩展性的核心库

name	unit tests	1ms async task	optional helpers	helpers	min js
yaku@0.13.7	✓	341ms / 108MB	✓	31	3.9KB
bluebird@3.3.4	x (28 failing)	291ms / 89MB	partial	100	52.2KB
es6-promise@3.1.2	x (27 failing)	509ms / 113MB	x	10	6.3KB
native@0.13.7	x (9 failing)	681ms / 168MB	x	13	0KB
core-js@2.2.1	x (4 failing)	910ms / 195MB	x	11	12.2KB
es6-shim@0.35.0	x (2 failing)	1055ms / 145MB	x	12	131.5KB
q@1.4.1	x (68 failing)	1594ms / 425MB	x	74	15.4KB

项目地址：github.com/ysmood/yaku

noflow

- 贯穿于项目各个部分的 http composer
- 基于 Promise，主动支持 ES7 async/await
- 轻量可控，无功能型依赖
- 项目地址：github.com/ysmood/noflow

```
import flow from "noflow";

let app = flow();

app.push(

    async ({ next }) => {
        await next();
        console.log("done");
    },

    $ => $.body = "hello world"

);

app.listen(8123);
```

junit

- 贯穿于项目各个部分的异步并发单测库
- 并发测试极大的减少了测试 API 的时间
- 项目地址：github.com/ysmood/junit

问答

谢谢